How to Set Up a High-Availability Database Cluster Using MySQL

Overview

High availability (HA) is essential for ensuring that critical databases remain accessible and resilient against failures. In this guide, we will walk through setting up a high-availability MySQL cluster that ensures your database stays online, even in the event of hardware failures. We'll use **MySQL Replication** and **MySQL Group Replication**, both of which allow for a scalable and fault-tolerant setup.

Prerequisites

- Three or more Linux-based servers (Ubuntu, CentOS, etc.)
- Root or sudo privileges on all servers
- MySQL 8.0 or later installed on all servers
- Network configuration allowing all servers to communicate with each other

Architecture

We will set up a **Master-Slave** replication cluster with **MySQL Group Replication** for fault tolerance. The cluster consists of:

1. **Primary Server (Master)** – Handles write operations and replicates data to secondary servers.
2. **Secondary Servers (Slaves)** – Replicate data from the primary server and handle read requests.
3. **MySQL Group Replication** – Ensures data consistency and automatic failover.

Step 1: Install MySQL on All Servers

Make sure MySQL is installed on all nodes in your cluster. On each server, run:

```
sudo apt update
sudo apt install mysql-server
```

After installation, check the status of MySQL:

```
sudo systemctl status mysql
```

Step 2: Configure MySQL for Replication

2.1. Edit MySQL Configuration

On each server, edit the MySQL configuration file (`/etc/mysql/mysql.conf.d/mysqld.cnf` or `/etc/my.cnf` depending on your system). You need to enable binary logging for replication.

```
[mysqld]

log-bin=mysql-bin

server-id=1         # Increment this for each server in your cluster

gtid-mode=ON        # Use Global Transaction Identifiers for better failover

enforce-gtid-consistency=ON
```

- **server-id**: A unique identifier for each MySQL server in the cluster. For example, set server-id=1 on the primary server, server-id=2 on the first slave, etc.
- **log-bin**: Enables binary logging, necessary for replication.
- **gtid-mode**: Allows for consistent replication across servers.
- **enforce-gtid-consistency**: Ensures GTID consistency across the servers.

2.2. Restart MySQL

After updating the configuration file, restart MySQL on each server:

```
sudo systemctl restart mysql
```

Step 3: Set Up Master-Slave Replication

3.1. Create Replication User on the Primary Server

Log into MySQL on the primary server:

```
mysql -u root -p
```

Create a replication user that will be used by the slave servers:

```
sql

CopyEdit

CREATE USER 'replica_user'@'%' IDENTIFIED BY 'password';

GRANT REPLICATION SLAVE ON *.* TO 'replica_user'@'%';

FLUSH PRIVILEGES;
```

3.2. Get Primary Server Binary Log Coordinates

Before setting up the slaves, take note of the binary log file name and position by running:

```
SHOW MASTER STATUS;
```

Example output:

```
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |   154    |              |                  |
+------------------+----------+--------------+------------------+
```

Make a note of the File and Position, as you will need these values when configuring the slave servers.

Step 4: Configure Slave Servers

On each slave server, log into MySQL and configure replication:

```
mysql -u root -p
```

Run the following command, replacing the values with the actual primary server's File and Position:

```
CHANGE MASTER TO
  MASTER_HOST='primary_server_ip',
  MASTER_USER='replica_user',
  MASTER_PASSWORD='password',
  MASTER_LOG_FILE='mysql-bin.000001',
  MASTER_LOG_POS=154;
```

Start the slave process:

```
START SLAVE;
```

Step 5: Verify Replication

Check if replication is running properly on each slave server:

```
SHOW SLAVE STATUS\G
```

You should see:

```
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

If both of these are Yes, replication is working.

Step 6: Set Up MySQL Group Replication

To enable **Group Replication** (which ensures automatic failover), you need to configure group replication settings on each node.

6.1. Edit MySQL Configuration on All Servers

Add the following to the `mysqld.cnf` file:

```
group_replication=ON

group_replication_group_name="my_group"

group_replication_start_on_boot=ON

group_replication_local_address="group_replication_address"  # Example: '10.0.0.1:33061'

group_replication_group_seeds="10.0.0.1:33061,10.0.0.2:33061,10.0.0.3:33061"
```

- **group_replication_group_name**: The name of the replication group.
- **group_replication_local_address**: The local IP address and port used for group communication.
- **group_replication_group_seeds**: The IPs and ports of the other nodes in the group.

6.2. Restart MySQL on All Servers

After saving the changes, restart MySQL on each server:

```
sudo systemctl restart mysql
```

6.3. Start Group Replication

Log into MySQL on each node and start the replication process:

```
START GROUP_REPLICATION;
```

Step 7: Monitor Cluster Health

Use the following command to check the status of the group replication:

```
SELECT * FROM performance_schema.replication_group_members;
```

This will show you the status of each server in the replication group.

Step 8: Handling Failover

In the event of a primary node failure, **MySQL Group Replication** will automatically promote one of the secondary servers to primary. You can also manually initiate failover by stopping the group replication on the failed primary and starting it on one of the secondary servers.

Conclusion

This setup provides a robust, high-availability MySQL environment with automatic failover and fault tolerance. By using **MySQL Replication** and **Group Replication**, you ensure that your database can withstand failures and continue to provide reliable access for your applications.